INTERNATIONAL BUSINESS MACHINES CORPORATION

P. O. BOX 218 RESEARCH CENTER PHONE WI 1-6500

YORKTOWN HEIGHTS, NEW YORK

MAY 19 1960

May 17, 1960

Memorandum To: Dr. Harwood Kolsky
438L Project Office
IBM Corporation
3102 Farnam Street
Omaha, Nebraska

Subject: RC-160

Enclosed is a copy of "A Description of Stretch." You will remember that I checked with you on the approximate timing formula. Thank you again.

L. R. Johnson (tw.)

L. R. Johnson
Dept. 475

LRJ/tw

Enclosure

A DESCRIPTION OF STRETCH

L. R. Johnson

ERRATA

page 17, figure at bottom of page, item 3: label is "address of VFL parameters".
should be "address or VFL parameters".

page 24, line 26: now reads "Non-destructive read takes 0.4 microseconds . . . ".
should read "Non-destructive read takes 0.2 microseconds . . . ".

page 32, line 6 forward: should read " . . . floating-point add, 1.0 microseconds; floating -point multiply, 1.8 microseconds; floating-point divide, 7.0 microseconds; and VFL addition on N bytes, (N + 2) (0.5) microseconds. (reference 19)"

Dec 1959

~~IBM CONFIDENTIAL~~

A DESCRIPTION OF STRETCH

L. R. Johnson

IBM

RESEARCH CENTER

INTERNATIONAL BUSINESS MACHINES CORPORATION
YORKTOWN HEIGHTS, NEW YORK

RC-160

A DESCRIPTION OF STRETCH *

by

L. R. Johnson

International Business Machines Corporation
Research Center
Yorktown Heights, New York

ABSTRACT; The organization of the Stretch computer system is described. Addressing, indexing, interruption, arithmetic and checking principles are outlined. Formats, indicators and instructions are surveyed as additional groundwork for discussion of the functional units, namely, the core storage units, the read-write devices, the basic exchange, the instruction units, the lookahead unit, the arithmetic unit, and the bus control unit. Twenty-one references are given.

---

* Prepared originally for the Computer Organization Series.

Research Report
December 10, 1959
RC-160

Printed in U. S. A.

TABLE OF CONTENTS

1. INTRODUCTION

Stretch is a high-performance computer under construction by IBM. Although the developmental effort has been traced back to 1954, it was in 1955 that the outlines of a ten megapulse computer were hammered into quantitative objectives for circuit development and for memory development. Preliminary work on the organization of the computer was underway in late 1955.

The first Stretch is scheduled for delivery in summer 1960. It is contracted for by the Atomic Energy Commission at Los Alamos. A second Stretch will be incorporated into the Harvest system that IBM is building for the Bureau of Ships. The subject of this report is the Los Alamos system; at this time, Stretch has not been announced as part of the IBM product line. Various engineering specifications are still subject to change.

The aim here is to report on the principal elements that constitute a Stretch, on the functions of each, and on the way in which the elements function together. The subject does not extend to logical design, much less to circuit design. The level of detail herein sought is that of "systems architecture."

The subject is approached through two intermediate levels of complexity. Sections 2 and 3 are offered primarily as groundwork for Section 4, "Functional Structure." Section 2 swings through basic ideas that underly the system. Section 3 surveys relevant formats and other general information. Some references are cited in context, but much of the material is too general for explicit documentation. All basic sources are given in Section 5, therefore, and associated with appropriate sections of the report.

2. GENERAL SURVEY

2.1 Design Tenets

The pattern of the single-address, one-instruction-counter type of computer is assumed in Stretch design. Some two-address instructions are found desirable, to be sure, and one-at-a-time execution is simulated, not observed. But the point of departure is clearly the IBM 704 and 705. Within this framework the Stretch system has brought new answers to many design problems. If some of these answers have a familiar look about them, it is because they have already influenced the IBM 709, 7090, 705-III and 7070.

By virtue of its hardware parameters, which are still impressive, Stretch would be a fast computer with any reasonable kind of organization. Delays of less than twenty millimicroseconds per level of logic are made practical with drift transistors. Core storage units of 16,384 words each are operated on a cycle time of two microseconds.

In setting the performance goals for Stretch, the targets for components were set high. Beyond this, however, considerable reliance was placed on the prospects that improved organization could enhance performance. These prospects were to include more efficient arithmetic processes, more powerful instructions, multiple memory units, and more overlapping in the instruction preparation and execution cycles.

It was in the early years of Stretch design, 1955 and 1956, that electronic digital computers really came into general acceptance. Field experience attested, widely and decisively, to the potential of general purpose computers. As it happened, the very same experience testified as well to unrealized potential - to potential lost because of inefficiencies inherent in the designs and practices of the time. Stretch might well be regarded as a response to the insights and the exuberance of that period.

In a decade of general purpose computers, Stretch is not only the fastest, it is in many ways the most general. It has an answer to all of the standard programmer grievances of the 1955-56 period. It has, for example,

- powerful indexing provisions,
- longer word length,
- floating point binary arithmetic,
- binary arithmetic on variable field length,
- decimal arithmetic on variable field length,
- logical operations on variable field length,
- separately-controlled read-write devices, and
- powerful interrupt provisions.

And it has speed. Simulation studies have estimated its potential for large scientific problems at sixty to a hundred times that of an IBM 704.

There is, throughout the Stretch design, evidence of much effort (and freedom) to generalize and to unify. This systems effort has achieved a certain elegance in the instruction set. With fewer than a hundred basic operation codes, and fewer than a dozen rules for options and parameter settings, a machine language of unusual richness is provided. To supplement and exploit this language, the hardware maintains sixty-four on-off status bits. These indicators can be interrogated for conditional branching and, better yet, many can be enabled to interrupt to corresponding exception routines.

Instruction definitions must be generalized to include possible indicator effects, of course. This implies additional detail for program control, as well as additional flexibility. In practice, the bulk of the indicator burden will be delegated to a supervisor program, with overriding responsibility elective on the part of the individual programmer. The result is a system with a high degree of convenience and flexibility from the programmer's viewpoint.

It is characteristic of the traditional stored-program computer that

(a) instructions fall, to some extent, into specialized categories, and

(b) each instruction execution is synthesized from a set of sequential tasks.

If generality is the first tenet in Stretch design, the second is to exploit the possibilities for parallelism inherent in (a) and (b).

Stretch instructions are taken to be of three kinds, viz., arithmetic, index arithmetic, and in-out. Each kind is executed out of a separate station, permitting a considerable degree of parallelism based on (a).

The life history of a typical arithmetic instruction execution in Stretch is fashioned out of such tasks as

- request memory access for instruction word,
- receive and check instruction word,
- fetch index word from index memory,
- modify instruction address,
- request memory access for operand,
- receive and check operand,
- perform arithmetic,
- check result,
- update addressable registers, and
- interrogate interrupt system.

This list is illustrative. To cover the complex arithmetic instructions, or the non-arithmetic instructions, it would have to be extended. It should, however, convey the idea of a possible parallelism based on (b).

At this point assume a number of hardware units for the sake of discussion: multiple memory units, busses, checkers, special-purpose registers, and an arithmetic unit. A task, as given in the preceding paragraph, generally involves more than one of these units. For each

task, therefore, assume a "procedure" that delegates appropriate work to hardware units. It is typical of a procedure to wait until it is free to go ahead, and then to delegate work to hardware units - with possible waits on busy units. This ability to proceed independently with a specialized mission will be termed "procedural asynchronism."

The Stretch design makes considerable use of procedural asynchronism to attain a high degree of parallelism based on (b). There are logical restrictions on the order in which things can happen, there are practical restrictions arising out of the allocation of tasks and hardware units, but there is no arbitrary machine cycle to which the timing of various tasks is subservient.

One comparison may be instructive. MIT's TX-2 computer, using separate memory units for the instructions and for the operands, attains a good deal of overlapping. Often it manages to overlap the operand memory cycle of one instruction with the instruction memory cycle of the instruction next to be executed. At best, however, the marginal time for one more arithmetic instruction execution is one memory cycle time. (It is more if the execution time has to be extended). The basic pattern for TX-2 arithmetic instructions is to overlap tasks onto a basic memory cycle, making relatively little use of procedural asynchronism.

By using multiple memory units, by buffering instructions, operands, and results for storage, and by using procedural asynchronism, Stretch can sustain arithmetic executions at rates well above that of the word rate for one memory unit. This ability dramatizes the extent to which Stretch designers have managed to circumvent the memory speed limitation, a limitation that has had a pervasive influence on Stretch design. The design considerations mentioned above, the instruction format, the binary addressing format - all are consistent with the need to minimize the word rate and to reduce system dependence on a basic memory cycle.

The procedural asynchronism mentioned above is indifferent to the design of the hardware units. The question of whether the hardware units do or do not employ asynchronous circuits is one that should be answered in the full context of efficiency versus estimated costs for design, production, maintenance, possible redesign, etc. A distinct advantage of procedural asynchronism is that such questions <u>can</u> be answered in context. (In Stretch, the decision has been to make considerable use of a master clock and of the synchronous circuit mode).

In general, systems involving multiple waiting times do not lend themselves to analytical expressions for the quantitative effects of structural or parametric changes, and simulation studies yield the only

realistic evidence on many design questions. Because the Stretch system does entail waits and bottlenecks, considerable use was made of simulation in its design.

A rather interesting rule of thumb is associated with the simulation studies in Stretch "design space." In a general purpose system with overlapping procedures, this rule says, no hardware unit should be busy all or nearly all of the time. If it is, procedures are too often lagging for it, and the performance of the whole system can be raised by the improvement of one unit. The optimum is said to be more like two-thirds usage.

2.2 Basic Formats

Sixty-four bits are available to the user in the Stretch word. Eight bits are appended for error checking and correction (ECC) functions. Within a word, bits are identified by location numbers 0 through 63.

Some instructions occupy a half word, others a full word. Index arithmetic, floating point arithmetic, and branch on indicator use a half word; variable field length (VFL) operations, in-out instructions, branch on bit, and a few others use a full word.

Sixteen words with locations 0 through 15 are defined as "addressable registers." These include the left half, right half, and the sign of the accumulator; the remainder in division; the multiplicand in multiply and add; a time clock; and the set of 64 indicators; and the address of an interrupt table. Locations 16 through 31 are assigned to sixteen index registers.

For floating point arithmetic, Stretch uses a 48-bit binary fraction, an 11-bit binary exponent, a fraction sign, an exponent sign, and three bits for flagging purposes. For the VFL operands in integer arithmetic and connective operations, a flexible format is used.

As an instance of Stretch generality, there is no machine-defined alphamerical code or collating sequence. No use is made of plug boards on the in-out devices. A card image, for instance, is obtained as 960 bits in fifteen consecutive memory locations (this is altered in an optional mode for including ECC bits with input). The case for program control is carried through to the operator's console, where unlabeled buttons, toggles and neons are correlated with memory bits - and made meaningful - through a "console-defining" routine.

2.3 System Schematic

Figure 1 depicts the principal elements in a Stretch system. On the left are six independent units of core storage. These are tied into two busses; one brings control information and store data to the units, the other carries data away from the units. Requests on the memory units are made via the bus control unit, which resolves conflicts among requests and allocates time periods on the busses. All data channels in the figure move 72 bits in parallel unless otherwise noted.

The memory bank is shared by

- the exchange,
- the high speed exchange,
- the instruction unit, and
- the lookahead unit,

each of these, in effect, constituting a special-purpose computer. The exchanges can request memory accesses for either storing or fetching. The instruction unit fetches only; the lookahead unit stores only.

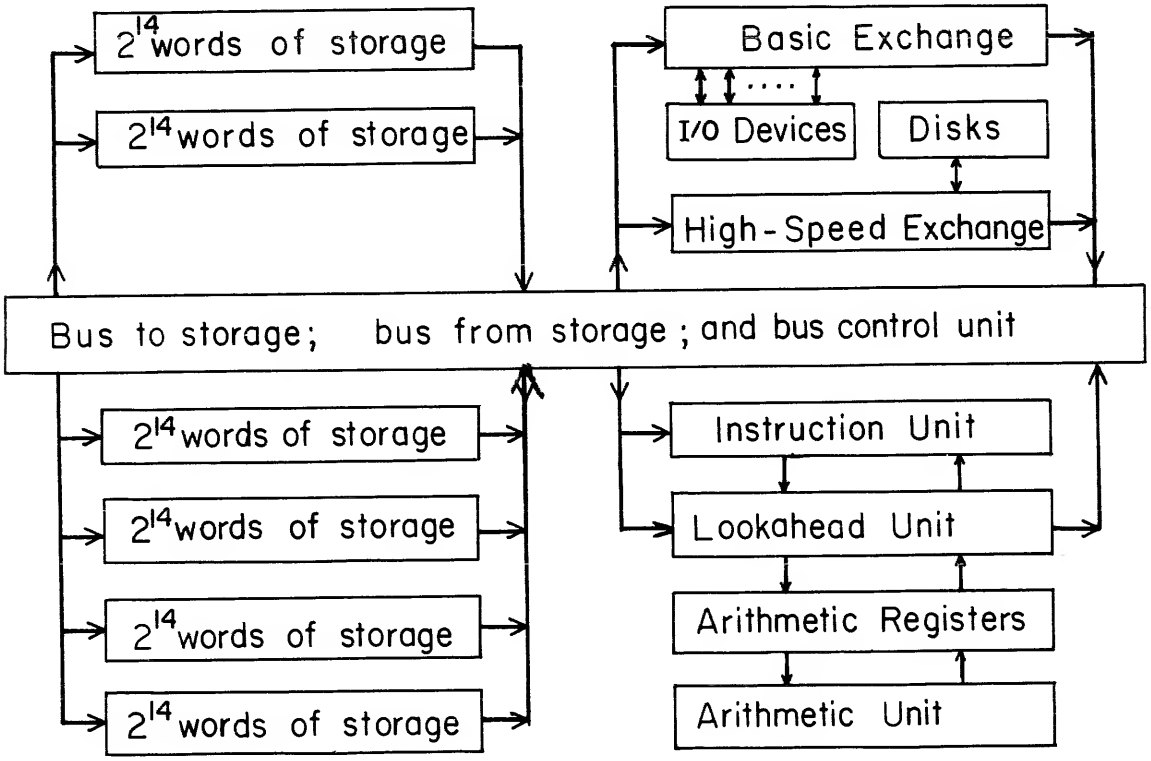


Figure 1 - Schematic of Elements and Data Flows in Los Alamos System

Up to 32 data channels can be accommodated by the exchange. Data channels are parallel for eight or for sixteen bits, as most suitable for the device involved. A device may use a whole channel, or several devices may share one channel through a control unit. Many channels can be in use at once. The maximum data rate for the exchange is one word every ten microseconds.

A high-speed exchange, with a four microsecond word rate, is provided for the very fast magnetic disk unit available with Stretch. Up to 32 units may be serviced, but only one can be reading or writing at any given time.

Taken together, the instruction unit, the lookahead unit, the arithmetic unit and registers constitute an ALU, i. e., an arithmetic and logical unit. (This grouping is denoted by "Sigma" in much of the Stretch literature).

The instruction unit performs what are sometimes called "red tape" operations. It has direct access to the index registers. Its first mission is to fetch instructions, index them, initiate requests for operands, and in general to arrange for the delivery of command-operand pairs to the lookahead unit. Its second mission is to intercept the index arithmetic instructions and execute them as they go by, even through this means doing them ahead of sequence.

The idea of sequence in Stretch must be associated with the addressable registers. The last instruction "executed" is the last instruction with the opportunity to modify addressable registers. Instructions executed ahead of turn in the instruction unit are blocked from such an opportunity until authorized by the lookahead unit.

The basic function of the lookahead unit is to maintain a backlog of work for the arithmetic unit, and thus to increase its average utilization. Lookahead has four levels of storage. Each will hold an operation code, an operand, and instruction counter reading, and necessary control bits for a simple half-word arithmetic instruction. Every executed instruction takes up at least one level in lookahead, although some take more. An in-out instruction, for instance, occupies one level until passed on to the exchange at the appropriate time. An index arithmetic instruction is most often represented by the old contents of the index register that it changed; if the sequence changes through conditional branch or interrupt, the index register can therefore be restored.

Lookahead is the only ALU unit that can store to memory. Many ALU control functions are assigned to it, such as timing the updating of addressable registers and timing of interrupts.

The arithmetic unit contains a 96-bit parallel adder for floating-point binary addition (true or complement); a smaller 12-bit adder for serial VFL operations; and additional circuitry for binary multiplication. Decimal multiplication and division are not implemented but can be obtained by automatic entrance to subroutines that use the instructions for radix conversion and binary arithmetic.

Associated with the data paths in Figure 1 are checking units: one in each exchange, one for the arithmetic unit, and one shared by the instruction unit and lookahead.

2.4 Addressing Principles

The binary radix is employed in addressing memory words and in-out devices. Each memory unit contains 2^{14} (16,384) words. The maximum number of memory units is 2^4 (16). The number of addressable bits in a word is 2^6 (64). Therefore, it takes 18 bits to address a word, 19 to address a half-word, and 24 to address a bit. The Stretch instruction format takes advantage of this fact and uses the fewest bits necessary for the addressing job to be done.

Each memory unit can be independently accessed. To exploit this fact, memory addresses are scattered over the units. In a system with more than four units, successive addresses are distributed among groups of four units. A system with six units, for example, has the first 2^{16} addresses distributed over four units, while the remainder of the addresses alternate between two units. For best performance in such a system, it is recommended that the two units contain instructions and the four units operands.

Addresses 0 through 31, the special and index registers, are addressable in general, but have some restrictions on their use.

Stretch instructions are single-address, for the most part. The address in the instruction is called the instruction address. This quantity is indexed to yield an effective address. Given an effective address, there are three modes in which it can be used:

(1) the immediate mode uses it as operand,
(2) the direct mode uses it to specify operand location, and
(3) the indirect mode uses it to specify the location of another effective address.

The instructions for index arithmetic make considerable use of (1). The typical mode of addressing is provided by (2). A special index loading

instruction provides (3) in a general way, and the principle is used in two special "execute" instructions.

2.5 Indexing Principles

Two formats are essential to any discussion of Stretch indexing. These are the index word and the control word, as shown in Figure 2.

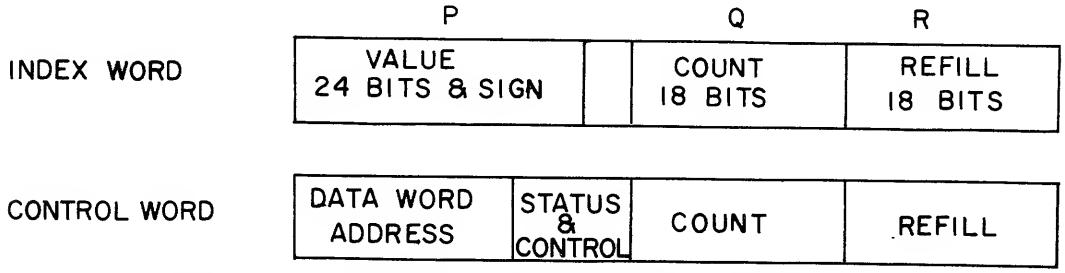


Figure 2 - Index and Control Word Patterns

Let P, Q and R denote the value, count and refill fields in an index word. When an instruction tags an index register, it is the signed value P that is added to the instruction address.

The generic instruction for Stretch index branching is, "Advance P, reduce Q by one, branch on Q condition, refill on zero Q." P may be advanced one word, decreased one word, advanced a half word, or unchanged. The branch can be on Q zero or Q non-zero. Other index arithmetic instructions provide for loading, storing, changing, and comparing the various fields P, Q and R. There is one instruction for summing P over a set of index registers. One provides indirect addressing in that a specified P is replaced by the field reached through a chain of load instructions. And there is a "rename" instruction with which any memory location can be made to serve the function of an index register.

A second kind of indexing, called progressive indexing, is available for VFL instructions. Here P is taken as the effective address of the instruction, and after the instruction is executed, P is increased (optionally decreased) by the instruction address. Q and R are treated as in direct indexing.

Some special addressing problems occur in any system where reading, processing and writing occur in parallel. The Stretch answer is the control word, with format such that it can also serve as an index word. A control word permits the exchange to read Q data words into an area starting at location P and then obtain another control word at location R. Writing is analogous. Skipping is possible. The conditions under which the operation terminates are encoded - by the exchange - in the status field of the control word.

Because of the consistency in control word and index word formats, one word of information can (a) serve to guide a record into memory (b) serve as an index quantity in processing the record, and (c) serve to guide an updated record out of memory. Records are effectively swapped by swapping their control words, which is a great help in arranging, merging, adding and deleting records.

2.6 Interruption Principles

As mentioned earlier, Stretch may be viewed as four limited-purpose computers sharing a common bank of core storage units. This memory sharing is accomplished by simple decision rules implemented in the bus control unit, and is in no way related to the stored program.

Turn now to the concept of "system sharing", a matter inherently related to program control. The end product of the system is a stream of executed instructions, and to share the system means to control this stream during various segments of time.

The pressures for system-sharing arise diversely. First, there is the technological fact that a Stretch has more processing potential, per dollar, than less expensive systems. Second, it is an observed fact that production runs on a computer must be supplemented and supported by program testing, machine maintenance, manual set-ups, and miscellaneous chores that use but a fraction of system potential. Third, there exist applications with requirements for immediate attention at disjoint intervals of time. Fourth, there are programming and operating efficiencies in allowing asynchronous in-out devices to interrupt upon completion of an operation. Fifth, there are programming and operating efficiencies in having exception conditions such as machine checks, overflows, and special flags cause interrupts.

The Stretch interrupt system is a generalized answer to the problem of system sharing. Each of 48 possible interrupt conditions may seize control - the incumbent program permitting. To each condition there corresponds a bit in the indicator register, and a request for interrupt is established by turning on the appropriate bit.

There are two kinds of program control over interruption. First, the interruption system may be "disabled" by instruction, and in this event no interruption can occur until an "enable" instruction follows. Second, 28 of the 48 indicators can be masked to prevent interruption. These two features provide a good deal of flexibility.

Interrupt priority is implicit in the location of bits within the indicator register; the leftmost "on" bit is first to be acted upon by the interrupt

system. The location of this bit within the indicator register is converted into a six-bit binary number, and this number is used to obtain an instruction word from a 48-word table.

The first instruction after interrupt is performed without disturbing the instruction counter. Unless this instruction effects a branch, the incumbent routine will continue as though nothing had happened. If the first instruction effects a branch, the interrupting routine must assume responsibility for the instruction counter and for any addressable registers and index registers that it changes. The first instruction of an interrupt routine will normally be "store instruction counter at y and branch," the last "branch to y."

Inasmuch as interrupts may occur within interrupts, it is likely that various system-sharing routines will need to have different interrupt tables. This generality is provided. An addressable register, "base word address," is used to locate the table, and the indicator number is used to find the word in the table.

2.7 Arithmetic Principles

With the current-switching circuits, complement representation can be obtained directly. One's-complement arithmetic with end-around carry is the natural choice for handling addition with unlike signs, therefore. When the main adder is operated as two independent adders, as is the case in divide, two's-complement arithmetic is used.

The parallel binary arithmetic operations are designed around three ideas. In the main adder, the method of "carry propagation" is used to reduce the levels of logic necessary for carry assimilation. A version of the "Sweeney" method, in which leftmost zeros in the partial remainder are shifted out, is used for division. The "carry-save" principle, through which carry assimilation can be deferred to the end of a series of carry-save additions, is employed to gain speed in multiplication.

Although carry assimilation is often viewed as a serial process for practical purposes, it need not be so viewed in the logical sense. It is possible to write a canonical expression for each sum digit as a function of the low-order carry and augend-addend digits. For an adder of typical size, the expression is hopelessly unwieldy, but by yielding some on levels of logic (hence, on time), the carry propagation method brings the general idea down into the realm of possibility. The Stretch adder is viewed as comprising five groups; in each group there are five sections; and in each section there are four bits. Various logical expressions are formed in parallel by section, then others in parallel by group, and then assimilated carries in parallel for the whole adder.

In ordinary binary non-restoring division, the dividend is diminished through a series of true or complement additions and one quotient digit is obtained for each addition. The Sweeney method starts with normalized dividend and divisor and takes advantage of the properties of leftmost zeros in the diminished dividend to realize more than one quotient digit per addition. In effect, some quotient digits come as a by-product of the act of renormalizing the dividend each time it is diminished. For random data and for quotients of ten or more bits, the method yields an average of 2.67 quotient digits per addition.

In Stretch this method is generalized to make use of the 0.75 and 1.5 multiples of the divisor. The divisor is shifted and added to itself in the upper half of the main adder to obtain these multiples. Their use has the effect of lengthening the strings that can be shifted over in normalization, giving an average of 3.7 quotient digits per addition (reference 7).

Before discussing Stretch multiplication, consider the following method for performing decimal multiplication with five multiples (true or complement) of the multiplicand. Even multiples will be chosen, viz., 2, 4, 6, 8, and 10. As an example, multiply 514 by 432. Our multiples are as follows:

	True	Ten's Complement
2	001028	998972
4	002056	997944
6	003084	996916
8	004112	995888
10	005140	994860

Let $\bar{x}$ denote the complement of digit x. Because 1 in the ten's position is equivalent to 10 in the units position, we can write

$$432 = 4\,(3 + 1)\,(2 - 10) = 44\bar{8}$$

and avoid the need for a three's multiple.

514		514
432		$44\bar{8}$
1028	or	995888
1542		02056
2056		2056
222048		222048

In Stretch multiplication, an octal representation is assumed. Even multiples are used, and are not stored but are gated as needed. Two, four and eight are obtained by shifting only. The main adder is used as in division to form a 1.5 multiple, and this is shifted to yield the six multiple. The octal plan reduces a 48-bit multiplication to sixteen carry-save adds followed by one ordinary addition. To further reduce elapsed time, multiplicand multiples for four octal digits are gated at each main step in the process.

Operations on exponents are performed in the serial arithmetic unit.

When a fraction is shifted left in normalization, the entering low-order digits may be specified as either zeros or ones. Provision is made for both unnormalized and normalized operations.

2.8 Checking Principles

There are conventional checks on the validity of memory addresses, channel addresses, operation codes, and the like.

All words moved to or from the main memory carry error checking and correcting (ECC) bits in the form of a modified Hamming code. These eight bits permit single-bit errors to be corrected and double-bit errors to be detected. Single-error correction and double-error detection on half words is done in the high-speed exchange using seven ECC bits.

The Hamming code is not used internally in the instruction unit, the lookahead unit, or the arithmetic unit, although the ECC bit-positions still exist for checking purposes. In floating-point arithmetic, operations on fractions are checked by similar operations on modulo-three residues. With this exception, the three units rely upon parity checks over various register fields and occasionally on duplicate circuitry.

There are four devices for generating and checking ECC bits: one each in the two exchanges, one in the arithmetic unit, and one shared by the instruction unit and the lookahead. These devices recognize the incoming format by context. They produce all relevant formats, and the requesting unit gates out the one desired.

The arithmetic registers have parity bits associated with them. Residues on the accumulator are maintained by the arithmetic checker. Residue on an operand is provided by the lookahead. Following a VFL operation, the contents of the accumulator are gated through the checker,

thus maintaining the residue and permitting VFL and floating-point instructions to be intermixed.

Memory accesses may be made in error because of machine malfunction or, as is more likely, because of program mistake. Program control over mistaken memory references is afforded by address monitoring. Two boundary registers, under program control, define an area of memory. References to memory include an address check against contents of these registers. An Interrupt indicator is set if the reference falls outside (optionally inside) the defined area. Separate indicators are provided for stores (#19), for data fetch (#20), and for instruction fetch (#21).

For maintenance purposes, about four thousand triggers are tapped through maintenance consoles. Visual exhibition is provided for static conditions, and a maintenance scanner is provided in an attempt to monitor the triggers under operating conditions. Upon detection of an error, the scanning system interrupts to record the status of all triggers. (The recording mechanism can be an ordinary card punch). About 2700 of the indicators concern the ALU, and in general mark the end of the execution in which the error occurred. Because the memory units, and particularly the exchanges, must complete relatively long operations, their indicators are latched as soon as possible after the error is discovered.

3. PROGRAMMING STRUCTURE

3.1 Addressable Register Formats

The first 32 memory locations are reserved for special-purpose registers. Of these, locations 16 through 31 are the index registers. Locations 0 through 15 are termed "addressable registers" and are listed in Table 1.

TABLE 1

ADDRESSABLE REGISTER FORMATS

Location	Name or Function	Bit Address
0	Permanent source of zeros	0 - 63
1	*Interval timer[1]	0 - 18
1	*Time clock (read only)	28 - 63
2	*Address of interrupt table	0 - 17
3	*Upper boundary	0 - 17
3	*Lower boundary	32 - 49
3	*Boundary control bit	57
4	Maintenance bits	0 - 63
5	Channel address (read only)	12 - 18
6	Signals for other computers	0 - 18
7	Left zeros count	17 - 23
7	All ones count	44 - 50
8	Left half of accumulator	0 - 63
9	Right half of accumulator	0 - 63
10	Accumulator sign byte	0 - 7
11	Indicators (0-19 read only)	0 - 63
12	Interrupt mask	20 - 47
13	Remainder in division (when defined)	0 - 63
14	Multiplicand in multiply and add	0 - 63
15	Transit register	0 - 63

*Area protected when interrupt system is enabled.
[1]Read-only except for Store P, Store Q and Store R instructions.

For measuring elapsed time over intervals of 8.5 minutes or less, the interval timer can be used. Time is counted with an oscillator that runs at 1,024 cycles per second, and therefore the reading from bit positions 0-8 can be taken as the approximate number of seconds left in an interval. The timer is stepped down at each pulse until it reaches zero, at which time Indicator #4 is set to interrupt.

As the interval timer is stepped down, the time clock is stepped up. The 36-bit field corresponds to about 777 days. Carry past the 36th bit position is ignored.

The boundary addresses are for monitoring of ALU memory accesses, and the boundary control bit tells whether alarms are to be for accesses inside or outside the defined area. "Maintenance bits" are not of programming interest. Attention request bits for up to 19 other computers can be defined in location 6; by turning such a bit "on", the stored program can signal the corresponding computer.

In VFL operations of the Boolean type, the "left zeros count" reveals the number of zeros to the left of the leftmost one bit in the result field. Meanwhile the "all ones count" stores the number of bits in the result field.

3.2 Data Formats

A floating-point number in memory has the format,

exponent ..... bits 0 through 10,
exponent sign... bit 11,
fraction ..... bits 12 through 59,
fraction sign ... bit 60, and
flags ..... bits 61, 62, 63.

The accumulator has two formats for floating-point operations, one for single precision and another for double precision.

	single	double
exponent bits	0 through 10	0 through 10
exponent sign bits	11	11
fraction bits	12 through 59	12 through 107

The fraction sign is always contained in bit four of a separate "sign byte" register of eight bits. This register also contains the flag fields. The term "double precision" is used for operations that involve a one-word memory operand and a 96-bit fraction in the accumulator. It takes two operations to store a 96-bit fraction.

Variable field length (VFL) operations take as operand a field of 64 bits or less. The field can start at any bit position in memory. The field is processed from the right, a "byte" at a time, using a two-word accumulator. In general, the programmer must specify

- length of operand field,
- accumulator bit against which rightmost bit of operand is to be aligned (accumulator offset), and
- some byte size.

Additional information is implied by a set of rules summarized in Table 2. (A byte is a set of bits processed at once, and the rules in the table constitute the only definition that is more meaningful). The programmer must also specify whether the operand is unsigned or signed; in the latter event, the rightmost operand byte will be treated as a sign byte.

TABLE 2

PARAMETER SPECIFICATION IN VFL INSTRUCTIONS

Parameter	Binary	Decimal	Boolean
Bits in operand byte[1]	8 implied	4 to 8 given	1 to 8 given
Bits in accumulator byte	8 implied	4 implied	8 implied
Size of operand field[2]	given	given	given
Bits in accumulator field[3]	128 implied	128 implied	8 per operand byte
Accumulator offset	given	given mult. of 4	given
Sign if operand unsigned	implied	implied	not applicable
Bits in sign byte, if any	given	operand byte	not applicable

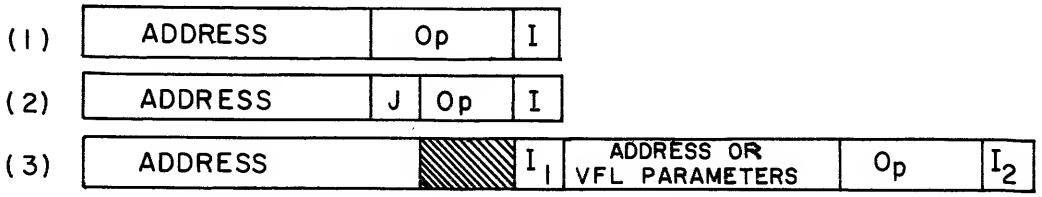
[1]With byte size over four in decimal, high-order bits are ignored to make an operand byte of four bits; with byte size given under eight in Boolean, high-order zeros are added to make an operand byte of eight bits.

[2]The operand field need not be an even multiple of the operand byte size; rules for the remainder bytes are defined.

[3]In binary and decimal arithmetic, the field length and offset number have the logical effect of unmasking one portion of an accumulator that acts as a unit of 128 bits.

3.3 Instruction Formats

Let I denote an index register to be used for address modification and J an index register that provides an operand or receives a result. Then the three patterns underlying Stretch instructions are



Pattern (1) is the prototype for floating-point arithmetic, (2) the prototype for index arithmetic, and (3) the model for branch on bit, in-out and VFL operations. Both sides of a type (3) instruction can be indexed.

3.4 Indicator Set

A listing of indicators is contained in Table 3. The general purpose of many of these will be fairly clear from the designation, but mention will be made of #5, #17, and #18.

TABLE 3

LIST OF INDICATORS*

Equipment Check
- 0 Machine check
- 1 Instruction check
- 2 Instruction reject
- 3 Exchange control reject

Attention Request
- 4 Time period has ended
- 5 Other computer wants attention

In-out Rejects
- 6 Exchange check
- 7 Unit not ready
- 8 Channel busy

In-out Status
- 9 Exchange program check
- 10 Unit check
- 11 End exception
- 12 End of operation
- 13 Channel signal
- 14 Reserved

Instruction Exception
- 15 Op code not valid
- 16 Address not valid
- 17 Unended sequence
- 18 Execute exception
- 19 Data store
- 20 Data fetch
- 21 Instruction fetch

Result Exception
- 22 Lost carry
- 23 Partial field
- 24 Zero divisor

Result Exception in Floating Point
- 25 Imaginary root
- 26 Lost significance
- 27 Preparatory shift over 48
- 28 Exponent $\leq 2^{11}$
- 29 $2^{10} \leq$ exponent $< 2^{11}$
- 30 $2^{8} \leq$ exponent $< 2^{10}$
- 31 $2^{5} \leq$ exponent $< 2^{8}$
- 32 $-2^{11} <$ exponent $\leq -2^{10}$
- 33 Exponent $\leq -2^{11}$
- 34 Remainder underflow

Flags
- 35 Data flag 1
- 36 Data flag 2
- 37 Data flag 3
- 38 Index flag

Automatic Subroutine Entrance
- 39 Loaded for binary op
- 40 Loaded for decimal op

Reserved
- 41 through 47

Index Result
- 48 Count zero
- 49 Value negative
- 50 Value zero
- 51 Value positive
- 52 Index low
- 53 Index equal
- 54 Index high

Arithmetic Result
- 55 To-memory operation
- 56 Result less than zero
- 57 Result zero
- 58 Result over zero
- 59 Result negative
- 60 Accumulator low
- 61 Accumulator equal
- 62 Accumulator high

Mode
- 63 Mode of Normalization

*Indicators 0-19 always interrupt, 20-47 interrupt unless masked, and 48-63 never interrupt.

Indicator #5 is activated when another computer wants attention; the identity of the interrupting computer will be stored in accord with some convention.

Indicator #17 is activated if an indirect addressing sequence continues through more than one full time interval of a millisecond. Indicator #18 reports that an instruction performed under control of an "execute" type instruction tried to branch but was suppressed. Indicators #19, #20 and #21 are set, as appropriate, when (1) the central processing unit attempts stores or fetches to a delimited area in memory and (2) the interrupt system is enabled. If the corresponding mask bit is one, the operation is suppressed and interruption occurs; if zero, the operation is completed.

3.5 Instruction Set

Exact formats, indicator settings, or unusual usages are not attempted in presenting the instruction list. Little use is made of symbols, although IC will denote "instruction counter" and X_i will denote the ith index register, where i runs from 0 to 15. The terms direct, indirect and immediate will have the meaning used in Section 2.4. The distinction between "different" instructions is more or less arbitrary in Stretch; conventions used by the designers are followed here. As presented, the instructions are separated by semicolons.

Index Arithmetic

Load X_i; Load value (direct, immediate plus or minus);
Load count (direct or immediate); Load refill (direct or immediate);
Store X; Store value; Store count; Store refill;
Add to value (direct, immediate plus, immediate minus);
Ditto and count; Ditto and count and refill on zero count;
Compare value (direct, immediate plus or immediate minus);
Compare count (direct or immediate);
Add to count (immediate plus or immediate minus);
Refill; Refill on count zero;
Load value with sum of values from specified X_i;
Load value by indirect addressing;
Load X_i ($i \neq 0$) saving memory address in X_0 but first store X_i in location given by contents of X_0;
Store value in address field of instruction with number of bits appropriate to the instruction;
Increment value, count and branch on count;
Increment value, count, branch on count, refill on count.

The value field of an index word takes up 24 bits, yet only 19 bits are available from the address of an immediate instruction. Simple rules govern the treatment of the five low-order bits in loading, adding or comparing value via immediate addressing.

In-out instructions. These are few in number and have but one variation. In typical usage, an in-out operation signals Indicator #12 (end of operation) when successfully completed. This potential interrupt can be avoided, if so desired, by using an instruction for the purpose. Let SEOP denote "suppress end of operation signal." All instructions occupy a full word and can therefore contain two addresses. In read or write, the first address specifies the channel and the second address specifies the location of the first control word. Data addresses and control information are provided by control words.

Read; Read SEOP; Write; Write SEOP;
Control (rewind, backspace, etc.); Control SEOP;
Locate unit or arc; Locate SEOP;
Release channel and clear status bits in control word;
Release SEOP; Obtain control word from exchange memory.

Branch and miscellaneous instructions. Let T denote an instruction that does not affect the interrupt mode or reset IC.

No operation; Branch; Branch relative to IC;
Enable interrupt mechanism and branch;
Enable interrupt mechanism, branch and wait;
Disable interrupt mechanism and branch;
Branch on indicator (off or on, leave or set to zero);
Branch on bit (off or on, leave or invert - leave or set to zero);
Store IC if suffixed half-word branched instruction is taken;
Move N words in memory; Swap n word-pairs in memory;
Store zero; Execute instruction T at location U;
Execute T as addressed by contents of location U, then augment U as though it were IC.

The last two instructions can use indirect addressing; i.e., they will chain until a non-execute instruction is found, or until a full millisecond time interval has elapsed (indicator #17 will then be set). The "no operation" code differs by only one bit from certain branch instructions.

Floating-point instructions. The numbers 48 and 96 are used to indicate the size of memory and accumulator fractions in those cases where a question might exist. All instructions are half word. Augment means "add but if accumulator goes negative clear it to zero!" Operations may be normalized or unnormalized as specified. Sign control is provided, typically as follows: signs unchanged, invert sign of memory operand,

assume operand positive, or assume operand negative.

Add; Add 48 to 96; Add to memory; Add exponent immediate;
Add 48 to 96 using accumulator exponent with addend;
Add operand exponent to accumulator exponent;
Augment; Augment 48 to 96; Augment to memory;
Load; Load with flag; Load cumulative multiplicand;
Reset 96 and load 48; Reset 96 and load 48 with flag;
Store; Store rounded; Store square root;
Store low 48 of 96 with adjusted exponent;
Compare; Compare with positive-signed accumulator;
Compare if accumulator-high indicator is on;
Compare with positive accumulator if high indicator is on;
Multiply 48 x 48 and save 48; Multiply 48 x 48 and save 96;
Multiply 48 x 48, save 96 and add to 96;
Divide accumulator 48 (extended with 48 zeros) by operand 48;
Divide as before but first swap accumulator and operand;
Divide 96 by 48 and save remainder; Shift 96.

With Indicator #63 on, an optional mode of arithmetic is used. Left shifts in normalization then bring in ones on the right, thus providing a basis for doing the same calculation two ways and estimating loss of significance from the two results.

Variable field length instructions. All VFL instructions occupy one word. The variable length data field may be unsigned or signed decimal or binary, or it may be any desired pattern of bits. The results of multiplications and divisions are aligned as though the operands were integers.

Several of the parameters necessary to a VFL instruction were given in Table 2. If the instruction is arithmetic, radix and sign control parameters must be given. If the instruction is a logical connective, a four-bit code is used to specify which of the sixteen possible logical connectives applies in combining memory and accumulator bits. The form of indexing must be specified as well. Ordinary indexing can be used with direct or immediate addressing. Progressive indexing can be chosen with six options for adding, counting and refilling.

Add; Add to memory; Augment; Augment to memory;
Count to memory; Load; Load with flag; Store; Store rounded;
Compare; Compare field; Compare if "equal" indicator on;
Compare field if "equal" indicator on;
Compare if "high" indicator on;
Compare field if "high" indicator is on;
Multiply; Multiply cumulative; Divide and save remainder;
Load cumulative multiplicand;
Memory to accumulator with radix change;
Memory to transit register with radix change;
Radix change in accumulator; Radix change in double accumulator;

Load transit and set indicator #39 or #40;
Perform logical connective with result to memory;
Perform logical connective with result to accumulator;
Perform logical connective just to set indicators and counts.

The last three instructions set the "all ones" counter and the "left zeros" counter. These counters are also used in the four VFL instructions that involve automatic entrance to subroutines: (1) multiply with decimal radix, (2) divide with decimal radix, (3) multiply and add with decimal radix, and (4) load transit register and turn on appropriate indicator. Each of these four conditions identifies itself by leaving information in the left zeros counter. Beyond this, (1), (2) and (3) simply place the specified offset parameter in the all ones counter and turn on Indicator #40 - thereby entering a common subroutine that determines what needs to be done.

Operation (4) is more general. It loads the transit register, places the given offset parameter in the all ones counter, and turns on Indicator #39 (binary) or #40 (decimal) in accordance with the given radix modifier. At this point the all ones counter contains information that can be used to distinguish among 128 subroutines for pseudo operations.

3.6 Time Estimation

There is no crisp, definitive way to estimate running time for Stretch. Here is a rule of thumb, however, that is useful for rough comparative and descriptive purposes. It is based on experience with a timing simulator (14), and is said to have a variation of perhaps ten percent from much more elaborate estimation procedures - for typical computational problems where the mix is quite high on floating-point instructions.

Let N denote the number of words involved in a move instruction, and the number of pairs involved in a swap instruction.

(1) Form a total of executions. Count moves 2N times, swaps 3N times, and other instructions once.

(2) Subtract branches not taken.

(3) Subtract index instructions that do not require operand fetch from main memory - if followed by VFL or floating point instructions.

(4) Multiply total by 1.4 microseconds.

(5) Add 5 microseconds for each interrupt, branch taken on indicator, or branch taken on bit.

A branch taken on count can be effected in the instruction unit with relatively-little delay for ALU as a whole. Branch taken on indicator or on bit, however, requires that lookahead be loaded from a new sequence of instructions. The time in (5) is associated with the delay involved in this reloading. Stretch performance is sensitive to the proportion of conditional branches, as will become clearer in the sequel.

3.7 Programming Aids

The aids to be described are designed primarily for the installation with a mix of batch processing jobs. Specifications are tentative.

STRAP, the symbolic programming system for Stretch, will provide the framework for mnemonic instruction formats, for pseudo operations, and for symbolic addresses. The first STRAP assembly system involves assembly processing on the IBM 704. This will be followed by an assembler that runs on Stretch. It is not planned that the 1960 version of STRAP will give the programmer debugging information back in the symbolic language, although this is contemplated for the future.

The increased processing speeds in Stretch spotlight the need for efficient read-write operations. Assume that the following could be estimated for a computer installation as a whole: (1) elapsed time necessary for all read-write operations with the exchange reasonably-well loaded, and (2) elapsed time for all ALU processing. Ideally, one would like to accomplish all work in the longest of these two elapsed times. Real-time considerations aside, multiprogramming in a Stretch-like system should be viewed as a step towards attainment of this goal. In reaching for higher efficiency, however, such multiprogramming involves additional cost. The nature - i.e., shape and generality - of the efficiency-cost curve is still a matter of speculation.

The second basic programming aid is a system for limited multiprogramming control. This system provides a convenient framework in which read-write chores can be run concurrently with a main program, along the lines that have been publicized for the IBM 7070. The idea is generalized to include an "automatic operator" through which various input (output) activities are queued and completed before (after) the corresponding main program processing. The objective here is to disassociate main runs as much as possible from all devices except high-density magnetic tapes, and to attain the objective without overwhelming the operators.

It is planned that input-output devices will remain symbolic in assembled programs. Symbolic will be converted to actual at running time. Devices will be loaded and unloaded upon instructions from the supervisor program.

The supervisor, version 1960, will permit temporary suspension of the main program and its tape devices to afford time for a debugging session. Repositioning of tapes may be optional later.

Fortran will be provided. A generalized multiprogramming scheme may or may not become justified. If so, the problem of storage allocation for a changing mix of active programs must be solved effectively. To aggravate this problem, Stretch suffers some loss of efficiency if the two-four (four-four, etc.) partition of instructions and operands is disregarded.

4. FUNCTIONAL STRUCTURE

4.1 Addressable and Index Registers

Because of the key role these registers play in the system as a whole, their functions have emerged earlier in the report.

Location 1, the interval timer and time clock, amounts to a seventeenth word in the index memory. Locations 2, 3, 5, 7, 8, 9, 10, 11 and 12 are transistorized circuits. Like all fast registers in ALU, the circuits use a variety of current-switching logic. Location 4 is implemented by switches on a maintenance console. Locations 13, 14, and 15 are simply locations in main memory.

The index memory, locations 16 through 31, is a parity-checked core storage unit with separate and independent read and write cycles. Non-destructive read takes 0.4 microseconds and write about twice as long.

4.2 Core Storage

Although the Los Alamos system has six, sixteen separate core storage units can be attached. Each unit is self-timed from the start of access, but is adjusted to synchronize with the slot system of the bus control unit. The memory unit farthest from the bus control unit is taken as a reference point and nearer units employ adjustable delay lines to simulate an equal distance.

Assuming no delay in service from the bus control unit or from the desired memory unit, the timing for an ALU fetch is as given below (a "slot" is 0.2 microseconds).

	Slots Needed	Slots Elapsed
Request to bus control (BCU)	1	1
BCU decodes; informs memory	1.5	2.5
Memory access	5	7.5
Memory returns data word	1.5	9
ALU checks data word	2	11

The requesting register in this case is notified of acceptance at the end of slot 2. In stores, acceptance provides a signal to start a two-slot data gate.

4.3 In-out Devices

New in-out devices can be added to Stretch by meeting the general requirements laid down by the exchange. The devices to be mentioned are illustrative.

The IBM 729-II and 729-IV magnetic tape devices are currently specified. To feed the exchange with 8-bit bytes, four 6-bit characters on tape are converted to three 8-bit bytes. Each tape control unit is assigned a channel, and up to eight devices can be tied to one control unit. A "locate" instruction is used to select the device of interest, while the channel is specified in the read or write instruction. Stretch words can be recorded as 64 or as 72-bit words, i.e., with or without ECC bits.

Currently specified are a 1000-cpm card reader and a 250-cpm card punch. Both use buffers of 960 bits. The buffers hold fifteen words in non-ECC mode. In ECC mode, the first 78 columns hold thirteen 72-bit words, and the last two columns are unused.

The electromechanical printer has 132 printing positions. Information to be printed is coded in an 8-bit code, with 16.5 words required for a full line. The information is moved from memory to buffer by the write instruction. The printing rate is 600 lines per minute for a 48-character set and 300 lines per minute for a 119-character set.

The operator's consoles, connected to the computer through the exchange, are viewed as in-out devices. The "central" console has two special keys, initial program load and emergency power off, and two unique visual indicators, running and inactive.

From any console, channel signal (indicator #13) can be activated. The console is designed with the equivalent of 192 bits (three words) of switch settings and 192 bits of visual information. A read operation is used to store the settings in three words, and a write operation uses three words to set up visual indicators. Additional information can be entered via keyboard and received via typewriter.

High-speed disk units are available with 2^{22} (over four million) words each. A unit is composed of 39 disks; 39 faces are used for even-numbered tracks, and the other 39 for odd-numbered tracks. Because each set of faces has a corresponding set of 39 read-write heads, and because consecutively numbered tracks do not fall in the same set, positioning of one track is possible while the preceding track is being read or written. Words are moved in parallel by 39-bit bytes, of which 32 bits are data. The other seven bits are ECC bits for single-error correction and double-error detection on half words.

Addressing is via 2^{12} locations called "arcs." Each arc contains 2^{10} words. Arcs are formed by dividing each of 2^{9} tracks into 2^{3} sectors. A locate instruction gives information for both track and sector. Time for a locate ranges up to 150 milliseconds. As the disks revolve every 34 milliseconds, a subsequent read or write will average about 17 milliseconds in finding the right starting sector. This 17 can be cut to around two milliseconds by reading full tracks - in which case it is possible for the system to start with the very next sector and adjust memory addresses to suit the particular choice of starting sector.

The transmission rate is one word every four microseconds. A special exchange is required for disks. Up to 32 disk units are said to be possible in one system. All units may be performing "locate" at the same time, but only one unit can read or write concurrently.

4.4 The Exchange Units

In addition to the basic exchange, a high-speed exchange is specified for the disk units. Its kindred design is simpler than the basic exchange, and will not be discussed here.

The end purpose of the exchange is to provide for, and regulate, the movement of data words between read-write devices and main memory. Typically, the exchange receives an instruction from the lookahead, carries out the execution, and then communicates back to the stored program. The exchange makes references to main memory to obtain new control words and to fetch or store data words. Via control units on its data channels, data bytes of eight or sixteen bits each are moved to or from in-out devices.

An exchange can have up to 32 channels. More than one device may be attached to a channel via a control unit, but only one device can be selected for a given channel at a given time. The exchange, therefore, sees only one device per channel. There is a possible exception to this rule: with a special adapter, one channel can be time-shared by up to 64 devices with very low data rates.

The exchange has a core-storage memory of 256 words with a one-microsecond cycle. The cycle leaves time for some processing, so that a word can be updated before being restored. Storage is assigned to 96 control words, 64 data words (two per channel) for ordinary channels, 64 data words for the possible time-shared channel, and 32 extra words. An 80-bit word length is used in the exchange memory: 64 bits for data, eight bits for ECC, and eight bits for exchange control purposes. The exchange is synchronized to its memory cycle, which is derived from a free-running ring. Instructions are executed by combining a number of different one-microsecond operations. Unbuffered devices are assigned numerical weights, in accordance with their data rates, and the stored program is expected to avoid exchange overloading by staying within a total weight limit. The prescribed limit is reached by eight IBM Type 729-IV tape devices each operating at 62,500 characters per second (and not using scatter-read or write).

The status of a channel is recorded in the status bits of its control word: exchange program check, unit check, end exception, end of operation, channel signal, suppress end of operation interrupt, and unit ready. The first five of these status bits have corresponding indicators in the indicator register (IR). When the exchange completes or otherwise terminates an operation for a given channel, it seeks to communicate with the stored program by sending the appropriate status bits to IR and the channel address to the channel address register. It is permitted to do this if one or more interrupts attendant upon the prior report have cleared the IR field to zero. Failing this, the exchange must wait. Because only one channel at a time can cause interruption, reports for other channels must be held up until the stored program clears the report it has. In-out status interrupts cannot be masked. End-of-operation interrupt can be avoided by a SEOP instruction, but this does not prevent interruptions from other conditions.

4.5 Instruction Unit

The first mission of the instruction unit is to provide the look-ahead unit with a stream of instruction-operand pairs. Four main functions are involved in the attainment of this objective:

- stepping the instruction counter,
- requesting and receiving instruction words,
- indexing instruction addresses, and
- requesting operands for delivery to lookahead.

The second mission of the instruction unit is to execute the index arithmetic instructions. There are secondary functions, of course, such as stepping the interval timer and the time clock every 1024 microseconds.

The unit has two registers Y_1 and Y_2 for main memory accesses, a register X for buffering direct accesses to the index memory, the instruction counter IC, and a register Z for use with the 24-bit algebraic adder.

The instruction unit has direct access to index registers for both fetch or store. It can fetch from main memory but must store to main memory via the lookahead unit. It has no direct access to addressable registers but must submit the request through lookahead.

The instruction unit, the lookahead unit, and the bus control unit are all synchronized to the same basic pulse source.

4.6 Lookahead Unit

The lookahead unit receives instructions from the instruction unit. It receives operands requested by the instruction unit. It transmits instructions and operands to the arithmetic unit; it transmits instructions and control word addresses to the exchanges. It receives operands to be stored from the instruction unit and from the arithmetic unit. It updates addressable registers. In event of interrupt, it restores index memory and authorizes an instruction counter (IC) value of record.

To perform its functions the lookahead has four levels of storage, two other important registers, and a sequence of five control counters through which each level must pass in disposing of an instruction. Lookahead also has a forwarding system by which operands can be moved from one level to another.

The following fields and information bits are associated with each level of lookahead storage: operation code, ten bits; operand, 64 bits; indicators affected by indexing, 12 bits; IC reading for next instruction, 19 bits; and status field, seven bits. Check bits are additional. Every instruction obtained by the instruction unit takes up at least one level in lookahead. A simple floating-point instruction takes one, but floating-point add-to-memory takes two. A VFL instruction takes one level for the instruction and parameters, and one or two additional levels according

as the field does not or does cross a word boundary. An index arithmetic instruction either saves a word (an index word it changed) in the operand field or provides a word for storage; it is also represented by the relevant indicator settings that correspond to its execution.

The "IC buffer" register contains the IC value for the instruction being executed. After execution is completed, this value is retained during interrogation of the program interrupt system.

The lookahead address register (LAAR) has three uses. (1) When a store instruction is loaded, LAAR receives the store address. (2) When an instruction specifies as operand the contents of an internal register, LAAR contains the register address. Lookahead can contain only one instruction of these kinds. (3) When LAAR is not used for (1) or (2), it contains the address of the last operand loaded into lookahead, and is said to be "not busy."

The instruction unit always checks LAAR before starting a memory fetch. If the instruction unit wants a word for itself (instruction or index arithmetic operand), and if LAAR is busy, an equal on compare is used to block the fetch until LAAR is no longer busy. If the instruction wants to request an operand for delivery to lookahead, an equal on compare means the word is in lookahead (3), will be in lookahead (1), or must be obtained by lookahead in exact sequence (2). This information is passed on to lookahead, where the forwarding system arranges data movements within lookahead.

If the instruction unit wants the contents of an addressable internal register, lookahead fetches and returns the data at the appropriate time.

4.7 Arithmetic Unit

The basic functions of the arithmetic unit are to perform (1) binary addition, multiplication and division on fixed-format fields, and (2) binary addition, decimal addition, and logical connectives on variable format fields. Central to the first function is a parallel binary adder with 102 bit positions, which will be termed the "main adder." Central to the second function is a parallel adder with twelve bit positions, supplemented by an eight-bit logic unit. For convenience, these are termed the "VFL adder."

All multiplication and division is done on binary radix. Binary VFL multiply and divide are so defined that they can be accomplished within the fixed-format facilities. Decimal multiplication and division make entrance to a stored-program subroutine, which will use instructions for radix conversion and binary arithmetic.

The VFL adder is used for exponent arithmetic in fixed-format operations. When the VFL adder is used for byte processing in VFL instructions, the main adder idles.

Four of the principal registers have an effective size of 64 bits each. These are register A, the upper accumulator; register B, the lower accumulator; register C, an operand register; and D, also an operand register. The accumulator as a whole is denoted AB. For VFL instructions it is convenient to think of CD as a continuous register, as there will be two operands if the field crosses a word boundary.

Closely associated with the main adder are register F and a shifter. F is used in multiplication and division and in general for shifting, which is accomplished through an adder-register-shifter loop. Shifts of six or less are possible in one cycle through the loop.

Among the special-purpose registers are the sign-byte register (8 bits); register G for multiply select signals (20 bits); register S for the sum from a carry-save add cycle (60 bits); and register T for the carry bits from a carry-save cycle.

The main adder is composed of 100 positions plus two high-order positions for true/complement control. The adder as a whole works with ones-complement arithmetic and end-around carry. Effective size for most purposes is 96 bits. Extra positions are needed in division, in which case the adder is divided, twos-complement arithmetic is used, and extra positions are needed for tapping of 3/4 and 3/2 multiples of the divisor.

In floating-point addition, the memory fraction from C and the accumulator fraction from AB are routed through the main adder, the checker, and back to AB.

In VFL byte operations, the operand or operands are in CD. The offset parameter is used to align the right edge of the field from CD with a position in AB. Starting from the right end of the respective fields, bytes of appropriate size are processed through the VFL adder in duplicate. Result bytes go back to AB. Upon completion of byte processing, the accumulator contents go through the checker to verify parity and update the residue-three information in the checker.

An octal radix is assumed in multiplication. Multiplicand multiples 2, 4, 6 and 8 - true or complement - are gated as needed. In the general algorithm, which was introduced in Section 2. 7, the multiple for an octal position assumes odd-even knowledge of the octal digit in the next higher position. The digit decoders therefore look at

the low-order bit in the next octal. An odd low-order octal is handled by the initial procedure.

A carry-save adder has three inputs and two outputs. At each step in Stretch multiplication, there are six inputs: multiples for four octal digits, and S and T from the prior step (these are zero at the first step). The six inputs are reduced to two with the network of adders in Figure 3.

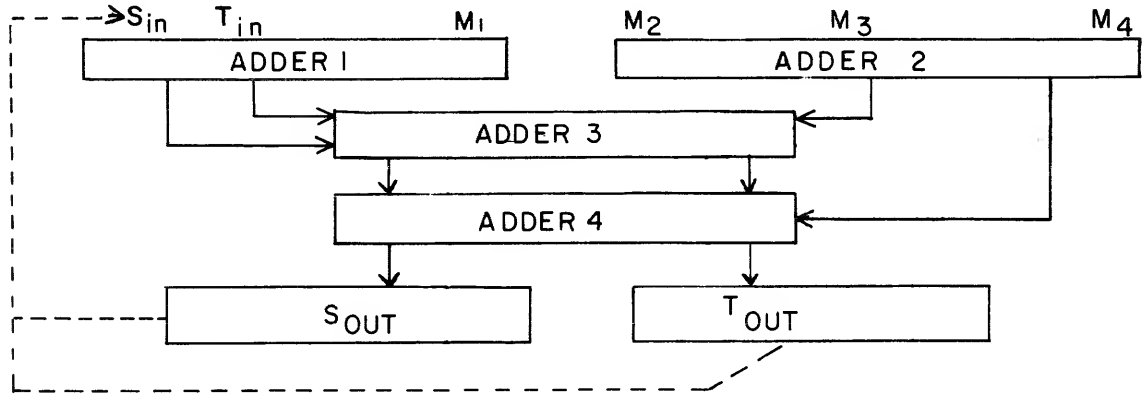


Figure 3 - Network of Carry-Save Adders

Three stages of carry-save addition are required to sum up the four multiples. This is interesting because the job could be done with four applications of one carry-save adder. The explanation: carry-save addition is very fast in itself but the return loop (dotted line) is relatively slow. The Stretch design is planned, therefore, to hold down the number of return loops.

At the start of multiplication, the multiplicand is in C and the multiplier moves from A into the high end of F. Four octal digits are decoded. Register C and the main adder supply multiples for the carry-save network. At the end of the cycle, the rightmost twelve bits in both S and T are stored in unused portions of F. After the cycle is repeated four times there are 60 sum bits in S and 36 in F, and there are 60 carry bits in T and 36 in F. The 96 sum bits and 96 carry bits go through the main adder for carry assimilation, back to F, through the arithmetic checker, and into the accumulator.

In division the lower order end of the main adder is used to form divisor multiples. The high end is used for addition. The dividend is placed in F and, as it is diminished at each cycle, is progressively shifted out of the high end. Quotient digits meanwhile shift into the low end of F. At the close, remainder and quotient are in F. These go to

the checker. The quotient returns to A. The remainder is treated according to the specific divide instruction.

The parallel add and basic multiply cycles in Stretch are not data-timed, although division is. Initialization procedures, particularly normalization, may introduce time variability. The following estimates are for typical conditions and include checking: floating-point add, 1.2 microseconds; floating-point multiply, 2.4 microseconds; floating-point divide, 7.5 microseconds; and VFL processing on N bytes, (N + 2) (0.6) microseconds.

4.8 Bus Control Unit

Two main busses, the memory-in bus and a memory-out bus, connect accessing registers to core storage. Each operates at a maximum rate of one word each 0.2 microseconds. Busses are under direction of the bus control unit (BCU).

The main BCU functions are to regulate the movement of (1) memory addresses, store-fetch commands, and data words from accessing registers to memory units, and (2) data words to receiving registers. BCU operates synchronously with the instruction unit and lookahead unit on the basis of 0.2 microsecond slots. The exchange operates asynchronously, and its requests are individually synchronized. Memory units are self-timed but are adjusted to synchronize accesses with BCU.

BCU selects accessing units according to a priority system and a memory unit as determined from the memory address. To afford a one-word-per-slot rate, control decisions overlap with address transmissions to memory units. During the move of one request to a memory, BCU is determining which request will be moved in the next slot. A request is honored if (a) the desired memory is available and (b) the requesting unit has higher priority than other units requesting available memories. Priority is in this order: basic exchange, high-speed exchange, lookahead, instruction unit. A data word to be stored is gated two slots later than the address; this time may be used in ALU to convert from an ALU checking code to the Hamming code used in memory.

In fetches, an output bus slot is reserved when the fetch request is moved to memory. The reservation is filed as a binary return address that specifies the register to read the data on the out bus. An ALU register is notified one slot before the data word is timed to reach the register, permitting it to request priority on the checker.

5. REFERENCES AND ACKNOWLEDGEMENTS

Although I did not encounter any bibliography of IBM confidential documents on Stretch, it should be mentioned that the IBM Technical Library, South Road Laboratory, Poughkeepsie, has on file two series of documents with relevance to machine organization. These are the Stretch Memo series with numbers 1 through 63 (1 November 1955 to 26 July 1957) and the Sigma Computer Memo series with numbers 1 through 33 (18 November 1957 to 6 July 1959). Over forty different authors contributed to the seven hundred pages under these titles. Although the memos vary greatly in scope and generality, they provide an incomplete but interesting record of analytical forces molding a machine organization.

For design objectives see Dunwell (1). Simulation studies were mentioned in Section 2.1 - for a report on studies using a timing simulator see Cocke and Kolsky (2). For amplification and analysis of points mentioned in Sections 2.4, 2.5 and 2.6, refer to Buchholz (3), Blaauw (4) and Brooks (5). For Section 2.7, see a paper by Brooks, et. al. (6) and three file memoranda (7, 8, 9) by Montgomery. None of these discuss carry propagate addition, however, and I have relied upon a paper by Weinberger and Smith (10) for the general ideas, which were worked out independently at the National Bureau of Standards. Section 2.8 is based largely on Bahnsen (11) and Stringfellow (12).

Sections 3.1 through 3.5 summarize from the Stretch information manual (13). The estimation procedure of Section 3.6 is not in print to my knowledge. Kolsky presented it to an internal IBM training course (14) and I have changed it slightly on the basis of a subsequent conversation with him. Section 4.7 is based on (14) and (15).

Use was made of Ulfsparre (16) for Sections 4.2 and 4.8; the manual (13) for 4.3; Stetler (17) for 4.6; and Fletcher, et. al. (18) for 4.4. In Section 4.7, use was made of (8) and (9). Revisions were made using Bloch (19), which has considerable material on the arithmetic unit.

For background on current-switching circuits, there is Yourke (20). For hardware counts and three basic Stretch circuits see Bloch (19). The classical article on error-correcting codes is Hamming (21).

While several people have been instrumental in the preparation of this report, I am particularly indebted to three. R. S. Ballance and E. I. Jordan have been helpful in providing references and system facts. Remarks by F. P. Brooks, Jr., first on the original outline and again on the first draft, enabled me to repair a number of weaknesses in technical emphasis and clarity.

(1) S. W. Dunwell, "Design Objectives of the IBM Stretch Computer," EJCC Proceedings, December 1956.

(2) John Cocke and Harwood G. Kolsky, "The Stretch Virtual Memory Concept and Timing Simulation Program," Poughkeepsie Product Planning Report P-17, 12 June 1959, 80 pages. (IBM Confidential)

(3) W. Buchholz, "Fingers or Fists? (The Choice of Decimal or Binary Representation)," TR 00.01011.683, IBM Product Development Lab., 17 April 1959, 21 pages.

(4) G. A. Blaauw, "Indexing and Control Word Techniques," IBM Journal of Research and Development, Vol. 3, No. 3, July 1959, pp. 288-301.

(5) F. P. Brooks, Jr., "A Program-Controlled Program Interruption System," TR 011.000.648, IBM Product Development Lab., 23 December 1957, 6 + 5 pages.

(6) F. P. Brooks, Jr., G. A. Blaauw, and W. Buchholz, "Processing Data in Bits and Pieces," TR 00.01000.674, IBM Product Development Lab., 27 January 1959, 18 pages.

(7) H. C. Montgomery, "Floating Point Division in SIGMA - The Algorithm Used," File Memo, 7030 Engineering Planning, 9 June 1959, 26 pages. (IBM Confidential)

(8) H. C. Montgomery, "Data Flow in Floating Point DIVIDE," File Memo, 7030 Engineering Planning, 30 June 1959, 5 + 1 pages. (IBM Confidential)

(9) H. C. Montgomery, "Floating Point Multiply in SIGMA," File Memo, 7030 Engineering Planning, 15 June 1959, 7 pages. (IBM Confidential)

(10) A. Weinberger and J. L. Smith, "A Logic for High-Speed Addition," NBS Circular 591, Section 1, 14 February 1948, pp. 3 - 12.

(11) R. J. Bahnsen, "Description of Checking in the Sigma System," Sigma Computer Memo #28, 9 February 1959, 7 + 4 pages. (IBM Confidential)

(12) W. R. Stringfellow, "Maintenance Scanner," File Memo, 7030 Maintenance Reliability, 29 September 1959, 5 + 2 pages. (IBM Confidential)

(13) STRETCH Data Processing System, Preliminary Manual of Operation, IBM Product Development Laboratory, 1 June 1959, with changes through 15 August 1959, 371 pages. (IBM Confidential)

(14) Stretch School #4218, IBM Department of Education, 16 February through 13 March 1959.

(15) Preliminary Stretch Programming Manual, IBM Applied Programming, 1 February 1959, 69 pages. (IBM Confidential)

(16) L. O. Ulfsparre, "Sigma Bus Control Unit," Sigma Computer Memo #20, 3 December 1958, 15 + 6 pages. (IBM Confidential)

(17) W. C. Stetler, "Lookahead Section of the Sigma Computer," Sigma Computer Memo #26, 4 February 1959, with later supplement, 27 + 5 pages. (IBM Confidential)

(18) R. P. Fletcher, G. E. Russell, F. C. Tung, and H. K. Wild, "Basic Exchange Machine Specifications," Exchange Memo #32, 4 December 1957, 100 pages. (IBM Confidential)

(19) Erich Bloch, "The Engineering Design of the Stretch Computer," TR 00.01000.702, IBM Product Development Lab., 1 December 1959, 31 pages.

(20) H. S. Yourke and E. J. Slobodzinski, "Millimicrosecond Transistor Current-Switching Circuits," WJCC Proceedings, February 1957 pp. 68-72.

(21) R. W. Hamming, "Error Detecting and Error Correcting Codes," The Bell System Technical Journal, Vol. 29, No. 2, April 1950, pp. 147-160.